BECAUSE

ISSUE 3
APRIL 1987

BULLETIN OF THE CANBERRA AMIGA USERS SOCIETY

(The Journal that explains the mysterious happenings inside your Amiga - BECAUSE)

MEETINGS:

Meetings are held on the second Wednesday of each month on the second floor of the J.G. Crawford building at ANU, starting at 8:00 PM. Roll up any time after 7.30.

The next meeting will be held on the 13th of May.
The following meeting will be on the 10th of June.

SUBSCRIPTIONS:

Annual fees are $20, payable at any of the monthly meetings to the treasurer.

AIMS/BENEFITS:

CAUSe is an independent group formed to bring together people who own, use or are interested in the Commodore Amiga computer. Members receive a bi-monthly bulletin providing a wealth of information concerning the Amiga and are encouraged to attend our monthly meetings.

COMMITTEE:

Treasurer: John Bishop --- 49 3786 (W) Editor: Craig Fisher - 54 6033 (H)
Secretary: Chris Townley - 41 3209 (H) Auxiliary: Peter McNeil - 54 2732 (H)
Auxiliary: Peter Whigham

BULLETIN BOARD:

CAUSE has it's own FIDO-NET bulletin board which is available free of charge to anyone with a modem.
The board is Baud rate auto-sensing and will accept 300/300, 1200/1200, 1200/75 and now also 2400/2400 baud rates.
Sysop: Mike Hurst-Meyers BBS: 59 1137 (24 hours)

IN THIS ISSUE:

News of the Amiga World

Craig Fisher

The first of the two most interesting things that have happened on the Amiga scene recently is that the price of the Amiga (now known as the Amiga 1000) has dropped significantly. The price has fallen from the $2495 at which it was released in Australia down to a shop price of about $1895. On top of this Commodore currently have a $300 cash-back offer (provided you send in some token piece of electronic equipment - such as the free calculator you got when you had your last film processed). That brings the price down to under $1600 which is quite incredible when you take into account the fact that the colour monitor that is thrown in with it is worth about $700. This makes the Amiga a fair bit cheaper than what was supposed to be the cheapest of the IBM-PC clones, the Amstrad 1512 PC, and as you know is a lot more sophisticated in both it's hardware and software. It used to be that the only advantage that the Atari ST had over the Amiga was it's low price - it has now lost that advantage. The Amiga has already 'made it' in the personal computer world and will clearly continue to establish it's position as the Mac has done.

The other significant piece of Amiga news is actually related to the first. It is the release by Commodore of the two new Amiga models - the Amiga 500 and the Amiga 2000. To fit these new models into their Amiga product line Commodore felt they had to drop the price of the Amiga 1000. After reading a few articles and talking to people about these two new machines I have found that I am not really impressed with either (they are of course being compared with the Amiga 1000). In case you haven't yet read or heard (apart from rumours) much about the Amiga 500 I'll give a brief description of it here. The Amiga 2000 is described in a bit more detail in an article elsewhere in this issue by John Kostakos.

The Amiga 500

Basically this new machine is an Amiga 1000 in a Commodore 128 type one-piece case (the keyboard is not detachable). This machine comes with one megabyte (1024K) of memory, twice the Amiga 1000's standard 512K, although it is supposed to be the bottom-of-the-line model. The main drawback with the 500 is that the great colour monitor is not included in the price of the package, which means it must either be purchased separately or you can connect it to a television and put up with the lesser picture quality. The other important feature of this computer is that Kickstart is now stored in ROM and need not be loaded from disk when the computer is turned on. Does this mean that Commodore-Amiga thinks that version 1.2 of the operating system is the final version - or do they intend to supply upgrades in chip form?

The Amiga 500 has one built in 3.5 inch disk drive which opens to the right hand side of the computer. The built-in keyboard has also undergone some changes. Keypad keys now also have IBM-PC type labels on them - e.g. Pg UP, Pg Dn, Home, End, Ins, Del. The DEL and HELP keys now sit out on their own, separate from the main keyboard section. The four arrow keys are also on their own and now in an inverted T configuration. Another nice addition to this model is the built-in real time clock so users will no longer have to set the date and time each time the computer is rebooted.

One feature that the new models of the Amiga were expected to have was upgraded custom chips which would be able to access more than the bottom 512K of memory. Both the new models (500 & 2000) still have the same custom chips but a photo of the Amiga 500's circuit board in AmigaWorld pointed out a new chip called "Gary" (the existing three are Agnus, Daphne and Paula).

The biggest setback for this new model could well be the fact that it has its expansion card edge on the left hand side (rather than right as on the 1000), and lower down. This means that add-ons designed for the Amiga 1000 plugged into the Amiga 500 will be facing the other way and the computer will have to be propped up. Either they got their plans back to front at some stage or they have a very good reason for doing this which isn't obvious.

The price of this new machine is expected to be about $1300, making it more expensive than the Amiga 1000 when you take into account the price of a good monitor, but then it does have twice as much memory. The Amiga 500 probably won't be available for a couple of months yet but when it is it will be interesting to see how many people will buy it instead of the 1000. It is expected that this model will be most attractive to and marketed more towards educational institutions.

DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/

Cheap Disks!!

CAUSe has available to members 3.5 inch disks at very good prices. Double sided **XIDEX** disks are available for the mere price of $42 per box of ten. Disks may be purchased at each monthly meeting or at other times during the week from John Bishop at the ANU (Ph: 493786). Nashua double sided disks are also available (when we can get them) for the same price. If necessary members may purchase disks directly from Nashua, (in the National Associations Centre, 71 Constitution Avenue, REID) but the discounts are not as great as when buying through the Society as we buy in bulk. Nashua's prices direct to members are $45 for double sided and $35.60 for single sided disks. Supplies of the **XIDEX** disks have been more dependable and are always for sale at meetings (until they run out).

It would be appreciated, although not necessary, if members wishing to purchase disks placed an order, along with the appropriate cash with John as the disks have to be paid for by us when they are picked up. This will also enable us to be sure of being able to sell what is bought.

FREE FREE FREE FREE

Yes, as an extra bonus - at no extra cost to you - you may like to have your disks smothered in **FREE** software. The Society has a large collection of public domain software which John has available at **ANU** for any members to copy.

DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/ DISKS/

AMIGA 2000 SERIES

By John Kostakos

The Amiga 2000 series is a new Amiga design by Commodore. Its objective was to provide the optimum design for the business or professional user. This meant providing flexibility to allow the user to configure the system to meet his or her requirements (and expanding needs), hence PC compatibility.

The A2000 is fully software compatible (Version 1.2) to the A1000 series although not all hardware add-ons are. Third party manufacturers designing RAM expansion boards, hard-disks, etc for the A2000 will also be designing such things as card cages for the A1000 to accept the A2000 expansion cards.

The standard A2000 has one internal 3.5" drive and one megabyte of RAM (expandable to 9 megabytes!). It also has a new keyboard layout with 96 keys which has the now-standard 'T' arrangement on the cursor keys and a full numeric keypad a la IBM. The machine has a battery-backed clock/calendar and Kickstart V1.2 in a 256 K ROM (does not have to be loaded off disk). Video resolution is the same as the A1000 ie. 320 x 256, 640 x 256, 320 x 512 and 640 x 512 - the latter two modes being interlaced. Serial and parallel ports are now also industry standard. All other ports remain unchanged with the exception of the 86 pin bus on the right-hand side of the A1000 which is now located internally.

Internally, the machine has 11 expansion slots with revised Amiga "Zorro" specification of 100 pins. Not all of these slots can be used on the standard system. Four of the slots are activated by installing the Bridgeboard which will leave three IBM-compatible slots free. Five of the other slots are general purpose and can be used for "fast RAM" expansion boards, fixed disk controllers, multi-function boards, etc. There are two other special purpose slots to be used for display interface boards (eg. Genlock) and for connecting cards directly to the 68000 bus (eg. "Turbo" boards or alternate processors).

The A2000 will support one to seven SCSI - pronounced "scuzzy" (Small Computer Systems Interface) devices through the hard-disk controller card. This also supports the standard ST506 (IBM type) devices. It is not necessary to buy two hard-disk drives if you wish to run one off AmigaDos and one off MS-DOS - one hard-disk can be partitioned for both DOS types.

For memory expansion, the A2000 has two type of cards. One is a 2 megabyte card which will support 1/2, 1 or 2 megabytes and the other is an eight meg card which can be populated as 4 or 8 megs of RAM. It is important to note that any of this expansion memory *cannot* be used by the Bridgeboard. The Bridgeboard itself contains 512 K of RAM. This can be expanded only by adding a PC compatible memory expansion card.

Having had quite a while "hands-on" on the A2000 recently, I can attest to the fact that it really is a brilliant machine, even if your BankCard does become very anorexic after your purchase. Price for the 'basic' A2000 is $A2995 *without* a monitor. All the expansion cards are well over $600 each. In other words for a fully expanded system including the Bridgeboard and other options, you can buy a <u>good</u> second-hand car! For the moment, I think my 1000 will suffice!

According to Commodore, we will see the A2000 "in about June". Hmmm....

WAITER! there's a grunge in my drive.

It should be of interest to many that you can actually save time by WAITing! Below I have two examples of startup routines, the first one takes 49.8 seconds and the second takes only 32.1 seconds.

Why is this so??

Well the answer lies in the disk access speed. In the first example when the run command has executed the command file executes the next command, the problem with this is that the program that is being run has not finished with the disk, so both tasks (the program and the next command) are trying to use the disk at the same time. The problem? The command is on track 1 and the program is on track 79! (well it sounds like it). If you try the first startup you find the disk drive going "grunge gurt grunge gurt grunge gurt grunge gurt grunge gurt grunge gurt grunge gurt" etc.

How to fix it?

The trick is to give the program enough time to load **then** execute the next command (bloody multitasking). The second startup sounds like this "grunge gurt gurt gurt gurt gurt...". I'm sure the second version is better for your drives life expectancy and it's QUICKER.

This startup-sequence goes Grunge...

```
echo "Workbench disk.  Release 1.2 version 33.46"
date ?
if EXISTS sys:system
    path sys:system add
endif
if EXISTS sys:utilities
    path sys:utilities add
endif
run clock
run popcli
BindDrivers
```

The Quick version of my startup-sequence.

```
echo "Workbench disk.  Release 1.2 version 33.46"
date ?
if EXISTS sys:system
    path sys:system add
endif
if EXISTS sys:utilities
    path sys:utilities add
endif
copy c/wait ram:
run clock
ram:wait 2
run popcli
ram:wait 2
BindDrivers
delete ram:wait
```

The Amiga Environment Part II (Devices I)

by Craig Fisher

This article is follows on from last issue's article which covered the Amiga libraries. Here I will begin a brief description of each of the Amiga devices which form part of the Amiga system software. The second half of this article will appear in the next issue of BeCAUSe.

Amiga devices are a really just a number of device drivers which make it a lot easier for programs to perform input or output to or from devices. Unlike programs on other computers such as the IBM PC type, Amiga applications need not know how to address the various devices themselves or even what type of devices are connected to the Amiga. Not all devices are external to the Amiga though - for example the audio device hardware is built in as standard. Although most of the devices are physical some are logical and exist only through software (e.g. the console, input, narrator and keyboard). The names of the devices contained in the Amiga system software are: audio, timer, trackdisk, console, input, keyboard, gameport, narrator, serial, parallel, printer and clipboard, all of which are documented in the book: "AMIGA ROM Kernel Reference Manual: Libraries and Devices". Routines for carrying out the input/output communication with these devices are actually provided by Exec. Information about using devices in general can be found in the Input/Output chapter of the "AMIGA ROM Kernel Reference Manual: Exec".

All the devices are accessed in more or less the same way. Before a device can be used it must be opened, by a simple function call to the (sensibly named) OpenDevice() function.

Each open device has an I/O request structure associated with it, which the calling program uses to communicate with the Device (by Device here I mean the software controlling the physical device). This structure is defined by the following 'C' code (from the 'Exec' manual):

```
    struct IOStdReq {
         struct    Message io_Message;
         struct    Device *io_Device;
         UWORD     io_Command;
         UBYTE     io_Flags;
         BYTE io_Error;
         ULONG     io_Actual;
         ULONG     io_Length;
         APTR io_Data;
         ULONG     io_Offset;
    }
```

Interaction with a device is carried out by calling an appropriate function using a (pointer to a) structure such as this as the parameter.

The fields in the above structure are used as below:

io_Actual is the actual number of bytes transferred during a device read or write

io_Length is the number of bytes requested to be transferred

io_Data is a pointer to the transfer data buffer

io_Offset indicates a byte offset for reading or writing to structured devices.

Exec provides four main routines for communicating with the devices, which are:

DoIO() - The most commonly used function. Initiates an I/O request and waits until it is completed. (synchronous)
SendIO() - Initiates an I/O request without waiting until it is completed. (asynchronous)
WaitIO() - Wait for the completion of a previously issued asynchronous I/O request (initiated by SendIO).
CheckIO() - Test whether an asynchronous I/O operation has been completed or not.

All of these routines use an I/O request structure as the only argument (actually given as a pointer to the structure). The request structure itself specifies which device the operation is to be performed on, what the operation is, and which unit of the devices to use. Devices are broken up into units - for example the trackdisk device, which handles the disk drives, is one device but can control up to four units. Another example is the Audio device - it also controls four units - the two left and two right audio channels. The Narrator device is an example of a device that has only one unit.

Well, onto a description of the individual devices...

The Audio device.
This device provides the ability to easily use one of the Amiga's most impressive features - it's high quality stereo sound. As mentioned earlier the Audio device controls four units - two left channels and two right channels. Basically this device allows you to tell it to play a waveform through a particular channel at a certain frequency and volume.

Actually it's quite a bit more complicated than that though - because the Amiga is multitasking you have to be allocated a channel before you can make noises through it. To be allocated one or more channels you issue the BeginIO command with the io_Command field of the IO structure set to ADCMD_ALLOCATE. All commands to the Audio device are carried out using a extension of the IORequest structure called IOAudio. When you issue a command to the Audio device you supply a priority level. If there is a free channel or your priority level is high enough to steal a channel from another process you will be allocated your requested channel (or channels). When you have finished using a channel you use the opposite command, ADCMD_FREE to free it so other processes may then use it if required.

To actually make sounds using the Audio device, the CMD_WRITE command is used. Fields of the IOAudio structure tell the Audio device the memory address where the waveform data starts, the length of the waveform in bytes, the period of the waveform (the length and period of a waveform determine it's frequency), the volume to play it at, and also the number of times to play it. After a CMD_WRITE has been performed it will finish when it gets to the end of the waveform (or has played the waveform the required number of times). If you wish to stop the sound being played before it has finished an ADCMD_FINISH command can be sent to the Audio Device. Various other commands are provided to control the playing of sounds and adjust sounds as they are being played.

The Timér device.

This device doesn't do a great deal, but what it does do can be very useful. It allows a program to go to sleep for a while and be signaled to wake up by the Timer Device in a specified amount of time (an especially handy feature in a multitasking machine such as the Amiga). This device also allows programs to find out what the system time is or set the system time. It also provides routines to add or subtract two times or to compare two times. A program could also use the timer device to perform some operation a certain number of times per second (or minute or whatever).

The Trackdisk device.

This device controls the disk drives and is normally only used by AmigaDOS. The Trackdisk device has commands to read or write sectors, turn the disk motor on or off, move the disk head to a specified track, format a track and to find out the status of the drive. When an application requests AmigaDOS to read or write a file it uses the Trackdisk device to read and write individual sectors from/to the disk. This device also has a number of units - each an individual disk drive.

The Trackdisk device also uses an extended form of IOStdReq structure which contains extra fields to store the number of times a disk has been changed (iotd_Count) and an optional pointer to sector labels in memory (iotd_SecLabel) to be written to or read from disk sectors. When a command is sent to the Trackdisk device the current count of the number of times the disk has been changed (TD_CHANGENUM) is passed in the structure.

If when the device receives the command the current disk-change number is greater to that in the request structure then the request will fail and an error (TDERR_DiskChange) is returned in the io_Error field of the I/O request block.

Before any read or write operation with the trackdisk device, the io_Data field of the request structure must point to (contain the memory address of) the area of memory to be used as the buffer to read into or write from. After the read or write, the io_Actual field will contain the number of bytes actually transferred. If the number actually transferred is not equal to the number requested then an error has occurred and the io_Error field of the IOStdReq structure will contain the number of the error. The include file for the trackdisk device, to be included by any program which uses it, defines the error numbers - e.g. for a disk write protected error the device would return the constant TDERR_WriteProt.

The Console device.

This device driver allows applications to perform simple keyboard input and screen (character) output. Applications can treat this device much like an enhanced ASCII terminal which recognises most ANSI command sequences as well as extra Amiga-specific command sequences. This device driver is one of the few that is not actually related to any physical device. Console device units are logical devices only; a new one is created each time the console device is opened.

The console device works in co-operation with Intuition (the Amiga user interface) - a console unit must be associated with an window already opened by Intuition. Although the console device doesn't do anything fancy from a user's point of view it seems to be one of the more complex and most often used devices. As with any of the other devices mentioned here this one is worthy of a whole article examining it in greater depth.

As with a terminal, interaction with a console device unit is done by sending and receiving character streams made up of single characters or special sequences. Normal printable characters (SPACE [20] -> ~ [126], HARD-SPACE [160] - [255]) sent to the console will be displayed in the window. The ROM Kernel Manual (RKM) also has a large list of command sequences accepted by the console for performing neat tricks such as BACKSPACE, LINEFEED, move up/down/left or right n characters, INSERT characters or lines, DELETE characters or lines, scroll lines up or down, change foreground and background colours, change text style, etc. Console command sequences all start with what is known as a CSI or Control Sequence Introducer which is actually the (hex) character 9B or alternatively ESCAPE followed by '['.

The RKM also defines a few routines to make console interaction easier (for the programmer). These include ConWrite to send a stream of characters (of specified length) to the console, ConPutChar to send a single character, ConPutStr to send a NULL terminated string of characters, QueueRead to queue a console read request, ConMayGetChar to get a character if one is ready and ConGetChar to get a character waiting if necessary. A very important concept associated with the console device is the keymap. The keymap is a table of values which contains an entry for each possible keycode (keypress), - i.e. a single key or a key combined with one or more qualifiers (SHIFT, ALT, CTRL, L-AMIGA, R-AMIGA). Keycodes have no relation to ASCII codes but they are used to look up the ASCII codes that they should return. Each entry contains either four one byte values representing: the character to be returned by that keycode, the key with one qualifier, the key with another qualifier and the key with both qualifiers; or alternatively: a pointer (four byte address) to a string, if the keycode is to return more than one character.

Keys such as the arrows, help and functions keys do not return a single character but a string of characters starting with the <CSI> character followed by one or two characters indicating the individual key pressed. For example if the HELP key was pressed, the program would receive from the console the following sequence of characters:

```
        <CSI>?~ or in hex ---> 9B 3F 7E
```

Normally when the console device receives a keycode from the keyboard device it converts it into the appropriate character or characters as in the keymap but the console may be requested to send on the raw keycodes to an application without converting them. A program can also request a number of other things from the console device such as a cursor position report, window bounds report or raw input events. If you ask to be told about raw input events then the console device will pass on to you information about disks being inserted or removed, preferences being changed, window resized or closed, menu selections, requester activity, timer events, pointer position or gadgets being pressed or released.

Stay tuned...to be continued (next issue)...

C Programming

Writing a Printer Driver

The Amiga Operating System implements Input Output comunication via software devices. These provide an I/O device independant standard for comunication. Each device includes fifteen routines which must be present in all devices plus any number of optional routines that may be device specific. Software devices include the trackdisk and printer devices.

When you wish to print something you open and write to the the "prt:" file. File I/O is performed through the trackdisk device while printer I/O is performed through the printer device. Opening the "prt:" file causes the printer device to be substituted for the trackdisk device. In both cases the interface is constant for at least the subset of fifteen routines.

The printer device provides a printer independant driver which allows programs on the amiga to control printer behaviour without being aware of the brand of printer that is actually connected. The printer device interprets a set of amiga defined command sequences into the appropriate commands for a specific printer via the ***printer driver*** selected by you in the preferences program. These command sequences are the same as those used by the console (examples include entering Italics or Bold Face mode). Printer drivers also provide features not available to the console such as graphics and page sizing. We shall investigate the writing of a printer driver further using the DataScape 3000 to demonstrate.

What we will need...

- ***a)*** Lattice C compiler V3.03 or greater (lower versions may work)
- ***b)*** Amiga Macro Assembler
- ***c)*** Alink (blink may work)
- ***d)*** An editor
- ***e)*** A printer and manual

The Driver

About the DataScape.

The DataScape is a 16 pin head, 20-136 column, tractor driven dot matrix printer. It includes 2 graphics modes (120 and 240 horizontal dots per inch). The vertical dot pitch is 120 dots per inch. The printer supports Gothic, Titan, OCRB, and Greek fonts available as draft, normal, bold, italics, superscript , subscript and elongated. There is no independantly selectable proportional font.

About the Driver.

What follows is the code to implement the DataScape driver. Your printer may be different from this. Where a feature is not supported by the DataScape I have left the field commented so that you can determine extras available to you.

There are several significant limitations to the printer driver standard command set as provided by the amiga. One such limitation is that there is NO horizontal microspacing available. This means that while you can move the print head vertically to a point you have no equivalent commands for the horizontal. If you need to, you can get around this by changing the horizontal character width value and spacing or backspacing as far as you wish to go.

Before we continue.

My C compiler lacked all of the essential files. If your copy also lacks them you enter them as below. The first missing file is Macros.i:

```
*******************************************
*
*  Macros.i ..... Printer Device Macros
*
*******************************************

*--- external definition macros

XREF_EXE    MACRO
   XREF        _LVO\1
            ENDM

XREF_GFX    MACRO
   XREF        _LVO\1
            ENDM

*--- Library dispatch macros

CALLEXE     MACRO
            CALLLIB   _LVO\1
            ENDM

LINKEXE     MACRO
            LINKLIB   _LVO\1,_SysBase
            ENDM

LINKGFX     MACRO
            LINKLIB   _LVO\1,_GfxBase
            ENDM
```

This file is an assembler include file so you do not have to do anything with it yourself. The compiler will use it when it needs to for the other assembly files.

The second file was pwait.c . This file/routine is used after printer initialisation to delay any transmissions until after the printer has got its act together. It is simply a timer device call for a set time out period (ie wait(TIME_OUT)), rather than actually wait on the printer proper. IN my case it was not a problem so a dummy routine that simply returned was used here. If you can't be bothered writing your own timer calls, a little counting loop with shifts and floating point divides should waste enough time.

Identify Your Printer.

The first thing we need to do is to make an assembly file that identifies and describes the printer to the device. This file is called printertag.asm Reproduced below is the DataScape printertag.asm file. Much of this file should be entered as written regardless of your printer.

You may need to change some of these fields. If is not a Black and White Graphics printer then your printer class will differ. The various classes are defined along with a lot of other useful stuff in the prtbase.i and prtbase.h files on your compiler disk. If yours is a colour printer then you are going to need more help than this article and I sugest you seek me out.

Other potential places of difference include horizontal and vertical dot pitches, the number of pins in the head (8 or 9 for epson look alikes for example) and the maximum number of columns to a line, and thus the max number of x pins. This last value is the maximum width of a line of graphics data measured in print columns for the graphics mode. In graphics mode, a column is considered to be one pin wide which is normally the width of the print head.

```
***********   Printer tag For Datascape **********
*
*   Printer device Dependent Code Tag
*
**************************************************


   SECTION      printer

*--- Include Files ----------

   INCLUDE "exec/types.i"
   INCLUDE "exec/nodes.i"
   INCLUDE "exec/strings.i"
   INCLUDE "devices/prtbase.i"

*--- Imported Names ---------

   XREF    _Init
   XREF    _Expunge
   XREF    _Open
   XREF    _Close
   XREF    _CommandTable
   XREF    _PrinterSegmentData
   XREF    _DoSpecial
   XREF    _Render

*--- Exported names ---------
```

```
   XDEF     _PEDData

*****************************

         MOVEQ    #0,D0          ; Show error for Open Library
         RTS
         DC.W     1            ; Version
         DC.W     1            ; Revision
_PEDData:
         DC.L     printerName
         DC.L     _Init
         DC.L     _Expunge
         DC.L     _Open
         DC.L     _Close
         DC.B     PPC_BWGFX      ; PrinterClass
         DC.B     PCC_BW         ; Colour Class
         DC.B     136            ; Max Columns
         DC.B     10            ; Num Char sets (this is a lie)
         DC.W     16            ; Num pins in print head
         DC.L     1440         ; Max X Pins (can't print past the 136'th col)
         DC.L     0            ; Max Y Pins (no limit to vertical height)
         DC.W     120            ; X Pins per inch
         DC.W     120            ; Y Pins per inch
         DC.L     _CommandTable   ; Comands
         DC.L     _DoSpecial
         DC.L     _Render
         DC.L     30

printerName:
         STRING <'DataScape'>

         END

============================================================================
```

We would have to assemble this file.

The Printer Command Set

The next step is to provide a table of printer specific commands. It is this table that the printer device looks up to convert Amiga command sequences into your printer command sequences. All the entries in the table represent trivial conversions. If a command is not implemented by the printer or the conversion is more complex than a simple conversion will allow (perhaps requiring flags to be set, or values calculated) place a string containing octal code 377 (ie "\377") in that location. In converting command sequences, the device investigates the CommandTable first, and then the functions in doSpecial.c. This gives you another bite at the cherry after the CommandTable

If for some reason you need a NULL character as part of a command string use the octal representation \376 which will be translated to NULL before printing.

The file should be called "data.c"

```
 ****** Datascape Command Table ***************/

char *CommandTable[]=
{
   "\377",                                  /* Reset */
   "\033\026I",                            /* Initialise */
   "\012",                                  /* lf */
   "\015\012",                               /* Cr, Lf */
   "\033\012",                               /* Rev. Lf */
   "\033@@0\033@A0",                      /* Norm Ch set */
   "\033@D1",                               /* Italics On */
   "\033@D0",                            /* Italics Off */
   "\033@E1",                                /* Underline On */
   "\033@E0",                            /* Underline Off */
   "\033@C1",                               /* Boldface On */
   "\033@C0",                            /* Boldface Off */
   "\377",                                  /* Set ForCol */
   "\377",                                  /* Set BackCol */
   "\033@B0",                            /* Normal Pitch */
   "\033@A0",                                  /* Elite On */
   "\033@A1",                                  /* Elite Off */
   "\033@A1",                                  /* Fine On */
   "\033@A0",                                  /* Fine Off */
   "\033@B1",                               /* Enlarged On */
   "\033@B0",                            /* Enlarged Off */
   "\377",                                  /* Shadow Print On */
   "\377",                                  /* Shadow Print Off */
   "\377",                                  /* DoubleStrike On */
   "\377",                                  /* DoubleStrike Off */
   "\033@G0",                                  /* NLQ On */
   "\033@G1",                                  /* NLQ Off */
   "\033@I1",                               /* Superscript On */
   "\033@I0",                            /* Superscript Off */
   "\033@H1",                               /* Subscript On */
   "\033@H0",                            /* Subscript Off */
   "\377",                                  /* Normalise */
   "\033D",                               /* PartLin Up as neghalf lf */
   "\033U",                               /* Partlin Down (half lf) */
   "\033@@0",                            /* US ch set */
   "\033@@3",                               /* French */
   "\033@@1",                               /* Germ */
   "\033@@2",                               /* Uk */
   "\033@@5",                               /* Danish 1 */
   "\033@@4",                               /* Swede */
   "\033@@7",                               /* Italy */
   "\033@@6",                               /* Spain */
   "\377",                                  /* Japan */
   "\033@@5",                               /* Norge */
   "\033@@5",                               /* Danish 2 */
   "\377",                                  /* Proport On */
   "\377",                                  /* Proport Off */
   "\377",                                  /* Proport Clear */
   "\377",                                       /* Set Prop Offset */
   "\377",                                       /* Auto left Just */
   "\377",                                       /*  "   Right "   */
   "\377",                                       /*  "   Full  "   */
   "\377",                                       /*  "   Just Off */
   "\377",                                       /* Place Holder */
   "\377",                                       /* Auto Center On */
   "\033@L16",                                    /* 1/8 Line Space */
   "\033@L20",                                    /* 1/6 Line Space */
   "\377",                                       /* Set form length */
   "\377",                                       /* Perf Skip On */
   "\377",                                       /* Perf SkiP */
   "\0339",                                    /* Left Margin Set*/
   "\0330",                                    /* Right Margin Set*/
   "\377",                                       /* Top  Margin Set*/
   "\377",                                       /* Bottom Margin Set*/
   "\377",                                       /* T&B Margin Set*/
   "\377",                                       /* L&R Margin Set */
   "\377",                                       /* Clear Margins */
   "\0331",                                    /* Set Horiz Tab */
   "\377",                                       /* Set Vert Tab */
   "\0338",                                    /* Clear Horiz Tab */
   "\0332",                                    /* Clear All Horiz Tab */
   "\377",                                       /* Clear Vert Tab */
   "\377",                                       /* Clear All Vert Tab */
   "\0332",                                    /* Clear All Hor&Vert Tab */
   "\033(09,17,25,33,41,49,57,65,73,81,89,97,A5,B3,C1,C9.",
                                             /* Set Default Horiz Tab */
   "\377"                                       /* Extended Commands*/
};
```

__Initialising the Printer.__

This is a standard file provided by Commodore Amiga. It should be entered exactly as written and assembled. Call the file init.asm.

Initialisation is a straight forward process of opening the usual libraries (Graphics and math) and declaring the nescessary base addresses.

```
*********** Init.asm for DataScape *************

   SECTION    printer

*--- Include Files -----------

   INCLUDE    "exec/types.i"
   INCLUDE    "exec/nodes.i"
   INCLUDE    "exec/lists.i"
   INCLUDE    "exec/memory.i"
   INCLUDE    "exec/ports.i"
   INCLUDE    "exec/libraries.i"
   INCLUDE    "macros.i"
```

```
*--- Imported Functions ------

   XREF_EXE      CloseLibrary
   XREF_EXE      OpenLibrary
   XREF        _AbsExecBase

   XREF        _PEDData
*--- Exported Globals --------

   XDEF         _Init
   XDEF        _Expunge
   XDEF        _Open
   XDEF        _Close
   XDEF        _PD
   XDEF        _PED
   XDEF        _SysBase
   XDEF        _GfxBase

******************************

   SECTION    printer,DATA
_PD       DC.L   0
_PED       DC.L   0
_SysBase   DC.L   0
_GfxBase   DC.L   0

******************************

   SECTION      printer,CODE
_Init:
        MOVE.L    4(A7),_PD
        LEA      _PEDData(PC),A0
        MOVE.L   A0,_PED
        MOVE.L   A6,-(A7)
        MOVE.L   _AbsExecBase,A6
        MOVE.L   A6,_SysBase

     ;----- open the graphics library

        LEA      GLName(PC),A1
        MOVEQ   #0,D0
        CALLEXE OpenLibrary
        MOVE.L    D0,_GfxBase
        BEQ    initGLerr

        MOVEQ   #0,D0
pdiRts:
        MOVE.L   (A7)+,A6
        RTS

initGLerr:
        MOVEQ   #-1,D0
        BRA.S   pdiRts
```

```
GLName:
        DC.B    'graphics.library'
        DC.B    0
        DS.W    0

*--- Expunge -------------

_Expunge:
        MOVE.L    _GfxBase,A1
        LINKEXE   CloseLibrary

*------------------------

_Open:
_Close:
        MOVEQ #0,D0
        RTS

        END
```

PIECES

BY COMMODORE AMIGA™

'SIM' PROFESSIONAL
MIDI INTERFACE
IN, OUT, THRU

ONLY $65

from:

PIECES
18 Darmody Street
WEETANGERA ACT 2614
(Dealer Enquiries Welcome)

Special Printer Functions

Some of the printer commands are too complex for a single string substitution to satisfy. Such commands are catered for in the dospecial.c file. We have reproduced the dospecial.c file for the DataScape below. Implenteation of such a file is largely straight forward. The cases for the switch statement may be found in devices/printer.h where all the legal Amiga printer commands are defined.

Note that there does not appear to be any documentation stating the maximum number of characters that may be placed in the outputbuffer. Experimentation indicates that when there are more than 45 characters (perhaps a little less, perhaps a little more), parts of the buffer are lost and/or the machine crashes. So be aware of that limitation.

Some initialisation parameters are found by examining the current preferences settings. Which have been copied earlier into the PD or printer data store.

```
 /****** DoSpecial for DataScape **********/


#include "exec/types.h"
#include "devices/printer.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;


#define HiNum( num ) ((num<100)?('0'+(num/10)):'A'+((num/10)-10))
#define LoNum( num ) ('0'+(num%10))

DoSpecial(command,outputBuffer,vline,currentVMI, crlfFlag,Parms)
```

```
char outputBuffer[];
UWORD *command;
BYTE *vline,*currentVMI,*crlfFlag;
UBYTE Parms[];
{
   int x=0, y=0;
   static char initMarg[]="\033@^000\033@^1D6";
   static char initFormLen[]="\033F00";
   static char initThisPrinter[]=
   "\033\032I\033@A0\033@G0\033@B0\033@C0\033@D0\033@H0\033@I0\033@L20";

/**********************************************************************
     This line has since been modified and only some of the below commands
implemented in the init string. REASON:  apparent buffer overflow. Below is
a list of  the sorts of  things  we  would  wish  to  include.  The numbers
represent the positions of these  commands  in  the string before some were
removed to make the program work.

Init  Printer,3Titan  Font,7Draft  Off,-11Elongated  off-Removed,15BoldFace
off, 19Italics Off,   22underline off,26Subscript  off,30Superscript off,34
6 LPInch (38+1)*/
**********************************************************************/

   switch(*command)
   {
      case aRIN    : /* Initialise Printer. */
         {

            do
            {
               outputBuffer[x]=initThisPrinter[x];
            }while(++x<36);

               /* Check if DRAFT */
            if((PD->pd_Preferences.PrintQuality)==DRAFT)
               outputBuffer[10]='1';
               /* Check Line Spacing */
            if((PD->pd_Preferences.PrintSpacing)==EIGHT_LPI)
            {
               *currentVMI=16;
               outputBuffer[34]='1';
               outputBuffer[35]='6';
            }
            else
               *currentVMI=20;
               /* CheckFont Pitch */
               /* Ignored as Not Emulatable */

               /* Set margins in Strructure */
            Parms[0]=(PD->pd_Preferences.PrintLeftMargin);
            Parms[1]=(PD->pd_Preferences.PrintRightMargin);
            *command=aSLRM;

         }
         /* Fall Through */
case aSLRM     : /* Set Left and Right Margins */
   {
      initMarg[4]=HiNum(Parms[0]);
      initMarg[5]=LoNum(Parms[0]);
      initMarg[10]=HiNum(Parms[1]);
      initMarg[11]=LoNum(Parms[1]);
      while(y<12)
         outputBuffer[x++]=initMarg[y++];
      return x;
   }
   break;
case aCAM     : /* ReSet Margins */
   {
      initMarg[4]='0';
      initMarg[5]='0';
      initMarg[10]='D';
      initMarg[11]='6';
      while(y<12)
         outputBuffer[x++]=initMarg[y++];
      return x;
   }

   break;
case aPLU     :
   *vline=(*vline<0)?0:1;
   break;
case aPLD     :
   *vline=(*vline>0)?0:-1;
   break;
case aSUS0     :
   *vline=0;
   break;
case aSUS1    :
   *vline=0;
   break;
case aSUS2     :
   *vline=1;
   break;
case aSUS3     :
   *vline=0;
   break;
case aSUS4     :
   *vline=(-1);
   break;
case aVERP0 : /* Number of vertical increments  for 8 lines/inch */
              /* Measured in printers vertical dot pitch. (see */
              /* printertag file for setting.                 */
   *currentVMI=16;
   break;
case aVERP1     :
              /* As above but for 6/lines per inch */
   *currentVMI=20;
   break;
```

```
       case aIND    : /* Set Line spacing */
          {
             outputBuffer[x++]='\033' ;
             outputBuffer[x++]='@' ;
             outputBuffer[x++]='L' ;
             outputBuffer[x++]=HiNum(*currentVMI) ;
             outputBuffer[x++]=LoNum(*currentVMI) ;
             return x;
          }
          break;
       case aSLPP   : /* Set Lines per page */
          {
             initFormLen[2]=HiNum(Parms[0]);
             initFormLen[3]=LoNum(Parms[0]);
             while(y<4)
                outputBuffer[y++]=initFormLen[x++];
             return x;
          }
          break;

       }
       return 0;
}
```

Printing Graphics

```
/********* Render.c For DataScape *************/

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "devices/prtbase.h"

extern struct PrinterData *PD;

/* For DataScape */

#define InBuf(a) PD->pd_PrintBuf[a]


/*************************************************************
     btohstr:
     Input  : bite - single int containing an 8 bit value in the
                     lower 8 bits.
     Output : Function returns a pointer to a character string.
                String has format:
                     "Render Status : xx\n"
                   where xx is replaced by hex value of bite.
     SideEffects : alters the contents of the string
                     referenced by rstr, a global string pointer.
     Purpose : Converts bite to a  to  character  hex  value and copies the
```

```
result into rstr. Returns rstr for printing during debugging.
     Bugs  & Restrictions : none
***************************************************************/

char *rstr;

char *btohstr(bite)
int bite;
{
   rstr="Render Status :  __ .\015\012";

   bite&=255;
   rstr[17]=(char)
(((bite/16)<10)?('0'+(int)(bite/16)):('A'+((int)(bite/16)-10)) );
   rstr[18]=(char)
(((bite%16)<10)?('0'+(int)(bite%16)):('A'+((int)(bite%16)-10)) );

   return rstr;
}

/*************************************************************
     Render :
     Input  : ct -
              x - current x (horizontal) position of pixel.
              y - current y (vertical) position of pixel
              status - render mode:
    [initialise, add pixel to line, print a line , clear & init the
     printer buffer, close & cleanup]

int Render(ct,x,y,Status)
UBYTE ct;
UWORD x, y;
UBYTE Status;
{
   static UWORD RowSize;
   static UWORD BufSize;
   static UWORD BufPtr;
   static UBYTE JumpTable[16]; /*Jump table for pixel position to col */
                               /* graphics conversion.               */
   BYTE (*WriteIt)();
   UWORD i;
   BYTE  err;


   WriteIt=PD->pd_PWrite;

   switch((Status&15))
   {
      case 0 :   /* Initialise */
 /****************************************************************
     Initialise Graphics Printing.
     The DataScape uses three bytes with the lower 6 carrying the
     print data and the high bit set to 1 for each column of 16 pins.
      For speed we use a lookup table table to determine where in the
     DataScape print column a given y pixel is. This is faster than
     the approach of shifting foreach pixel used in the standard Amiga
     printer drivers as we do the shifting once at initialisation rather
     than for each pixel.
      Make a pixel to print pin jump table.
 ****************************************************************/
         for(i=0;i<16;++i)
            JumpTable[i]=( (i<6)?(1<<i):
                    ((i<12)?(1<<(i-6)):(1<<(i-12)) ) );
/*  At init. the x parameter is the max width of the line. */
         RowSize = x;

/*******************************************************************
   Buffer size : allow 3 bytes for Graphics mode print command sequence
              + 1 for the end of transmition character (epsons don't
              have this) + 3 * the row size for graphics data.
*******************************************************************/
         BufSize = 4 + (RowSize * 3);

/* Allocate 2 Buffers for greater printing speed */
         if(!(PD->pd_PrintBuf= (UBYTE *) AllocMem(BufSize*2,MEMF_PUBLIC)))
            return -1;

 /***************************************************************
     Initialise Printer
     If you wish to initialise the printer before graphics
     printing you would do so here. You would need a working
     pwait routine. The calls look like this:
          if(err=(*WriteIt)("\033\032I",4))return err;
         if(err=PWait(1,0)) return err;
*****************************************************************/

               /* Set Line Spacing 16/120 */
         if(err=(*WriteIt)("\033@L16",6)) return (int)err;
         BufPtr = 0;
         return 0;
         break;

      case 1 :   /* Put Pixel in Buffer */
/*******************************************************************
     The printer device sends one printer pixel at a time to the printer
buffer. The position is determined by the x and y co-ordinates. We use
the jump table to determine where the pixel should go in the buffer.
*******************************************************************/
         {
            y&=15;
            InBuf( (BufPtr+(x*3)+3) + ((y<6)?0:((y<12)?1:2))
                              )|=JumpTable[y];
         }
         return 0;
         break;

      case 2 :   /*Print Buffer */
         if(err=(*WriteIt)(&InBuf(BufPtr),BufSize))
```

```
              return (int) err;
           (*WriteIt)("\015\012",3);
           BufPtr = BufSize - BufPtr;
           return 0;
           break;

        case 3 :   /* Clear and Initialise Buffer */
           {

             UBYTE *ClrPtr, *EndPtr;

              for(EndPtr=(ClrPtr= &InBuf(BufPtr))+BufSize; (++ClrPtr)<EndPtr;
                       *ClrPtr = 64 );
              InBuf(BufPtr)=27;
              InBuf(BufPtr+1)='@';
              InBuf(BufPtr+2)='J';
              InBuf((BufPtr+(BufSize-1)))=4;
              return 0;
           }
           break;

        case 4 :   /* Free   PrintBuf Mem */
                 /* Initialise Printer */
           {
              BYTE (*xBothReady)();

              xBothReady=PD->pd_PBothReady;
              if(!(err=(*WriteIt)("\033\032I",4)))
                 err=(*xBothReady)();
              FreeMem(PD->pd_PrintBuf,BufSize*2);
              return (int) err;
           }
           break;
        default :
           return 0;
     }
}
```

The graphics driver resolution is effectively limited by the Amiga screen resolution. Thus the above driver will give a printout similar to the epson driver when you use a straight screen dump. There are a number of ways to improve this. One such technique involves specifying the printer as being the same as the screen resolution and then smoothing between the pixels to achieve a less chunky image. Unfortunately space prevents me from expanding on this now. Perhaps we can look at this topic in a later issue.

Assembling, Compiling and Linking

The assembler must be run with -c w180000 to increase the workspace size for the printertag.asm assembly. This is because of the number of assembler include files used by this file. Add to your s directory the following file called "ass" :

```
ASSEM:c/assem -c w180000 -i ASSEM:include <file$t1>.asm -o <file$t1>.o
```

(The assembler line should not be split as above). Ass is used thus:

```
    1> execute ass my_assembler_file
```

My_assembler_file is replaced with the assembler file (extension .asm) that you wish to assemble.

All c files must be compiled with -v on the LC2 pass to prevent unresolved cross references at link time. Add to your s directory the following file called "lcp" :

```
LC:lc1 -iINCLUDE: -iINCLUDE:lattice/ <file$t1>
LC:lc2 -v <file$t1>
```

It is used thus:

```
    1> execute lcp my_c_file
```

My_c_file is replaced with a c file to compile (extension .c).

Finally linking is performed with the following batch file. Add this file to your s directory calling it "linkp" :

```
link:alink from printertag.o+init.o+data.o+dospecial.o+render.o+pwait.o to
<file$t1> lib LIB:lc.lib+LIB:amiga.lib
```

In the above file, the link line is intended to be on one line rather than three. Otherwise, it will not work. Linkp is used thus:

```
    1> linkp My_printer_name
```

My_printer_name is replaced by the name by which you wish to refer to your printer driver. In my case it was DataScape.

<u>Loading your Printer Driver.</u>

Having made a printer driver, it must be copied to the devs:printers directory of you WorkBench disk. The driver id then available to you by running preferences and configuring the printer device to your Driver. Don't forget to check the other printer settings in preferences to ensure they reflect the prefered defaults for your new driver.

Preferences printer selection differs under 1.2 from 1.1. Under 1.1 you select Custom from the range of printers offered and then type the name of your printer driver in the string gadget below it. Under 1.2 your driver will have been added to the range of printer drivers offered and you simply select it.

This issue's dose of hints 'n stuff

Craig Fisher

Yet another useful but little known fact:

V1.2 Requesters

(This feature was introduced with the version 1.2 release of the Amiga Operating System.)

Whenever a requester rudely pops up in front of you and your hands are at the keyboard it is (was) a nuisance to have to move a hand to the mouse to click on either the left or right (or only) gadget. Well now you can imitate that using the keyboard:

pressing **<LEFT-AMIGA><V>** is equivalent to pressing the left gadget
and pressing **<LEFT-AMIGA><B>** is equivalent to pressing the right gadget

These keys work in the present release of the Operating System but are apparently subject to change.

Amiga Resources

Does anybody know anything about Resources on the Amiga?

The ROM Kernel Manual lists three routines associated with Resources:

AddResource, *OpenResource* and *RemResource*

but the only information it gives about them is that they add a resource to the system, open a resource or remove a resource - not very helpful. They seem to be some sort of hybrid library or something. They don't seem to be very well defined or documented yet and maybe not even implemented(?). Maybe they are a possible future feature or an old, redundant one. Anyone know anything more about them?

NEWCLI zap

How to change the default window size for NEWCLI.

If you don't like the shape and location that NEWCLI brings up by default you can always use something like:

"NEWCLI /300/200/76/45/MY_WINDOW"

- that is if you can remember all that and then type it properly. An easier way to make NEWCLI bring up the sort of window you usually want is to change the NEWCLI program itself. To do this you will need the very handy (and Public Domain) program called FileZap (reviewed in the December '86 issue). If you FileZap NewCli and then look through the file you will find a piece of text describing the size and location of the window to open - something like:

"CON: /233/24/21/43/NewCLI"

You can change the numbers in this string to set the left, top, width and height of the window to suit yourself. You can also change the name of the window to something much more interesting. The only restriction is that you must not change any characters past the end or beginning of this piece of text.

Proportional Gagdet colours.

At last I have found out how to set the colours of proportional gadgets. Gadgets are normally drawn when a window is opened. The Window structure has fields which store the colours of the foreground and background pens. When drawing the imagery for the proportional gadgets Intuition looks at these fields and uses the pen colours found here for the body and the knob of the proportional gadget. These fields of the Window structure can be change by using the SetAPen, and SetOPen functions, which should then take effect on gadgets added (and Refreshed) after the window is opened.

The Query command.

One of the pre-release versions of V1.2 floating around had a new command in it's C directory called QUERY - some of you have probably come across this and wondered what it was all about. This looked like a very (potentially) useful command but there was no documentation to be found on how to use it anywhere. After a bit of fiddling around with it I worked out how to use it. The command actually lets you create a requester with left and right gadgets from CLI (i.e. usually in a script file) and get the user's response to it. The command has the following syntax:

```
QUERY "Main question text" "Left text" "Right Text"
```

which will bring up a requester displaying the *main question text* and two gadgets each with accompanying text. The user then clicks on on of the two gadgets and control will return to the script file. (That was the easy bit - the hard bit was finding out how to check what the response was.) To check whether the user's response was positive or not (the left gadget is always the positive one) the something like the following is used:

```
QUERY "Are you happy?" "Yes" "No"
IF ERROR
SKIP NEG_CHOICE
:
:          {commands for positive choice here}
:
ENDIF
SKIP ENDFILE
LAB NEG_CHOICE
:
:          {commands for negative choice here}
:
LAB ENDFILE
```

Using this you could do things such as create a Startup-Sequence file which asks you whether you want to use the CLI or WORKBENCH and if you want to use the CLI then whether you want a RAM disk created or not and act accordingly.

KickStart in ROM

Steve's now have kits available which enable you to have KickStart V1.2 permanently in ROM rather than having to load it from disk each time you turn on your Amiga. Installing these ROM chips also frees up 256k of RAM, giving you a 768k Amiga. The price for this is about $245.

Advertising in BeCAUSe

BeCAUSe is willing to accept advertising from both members and businesses. Members may submit small (less than 10 lines) classified advertisments to appear in BeCAUSe for free. Advertising rates for businesses are $30/page, $15 for half a page, $10 for a third and so on.

If interested in advertising in BeCAUSe contact the editor, Craig Fisher on 54 6033 (AH).

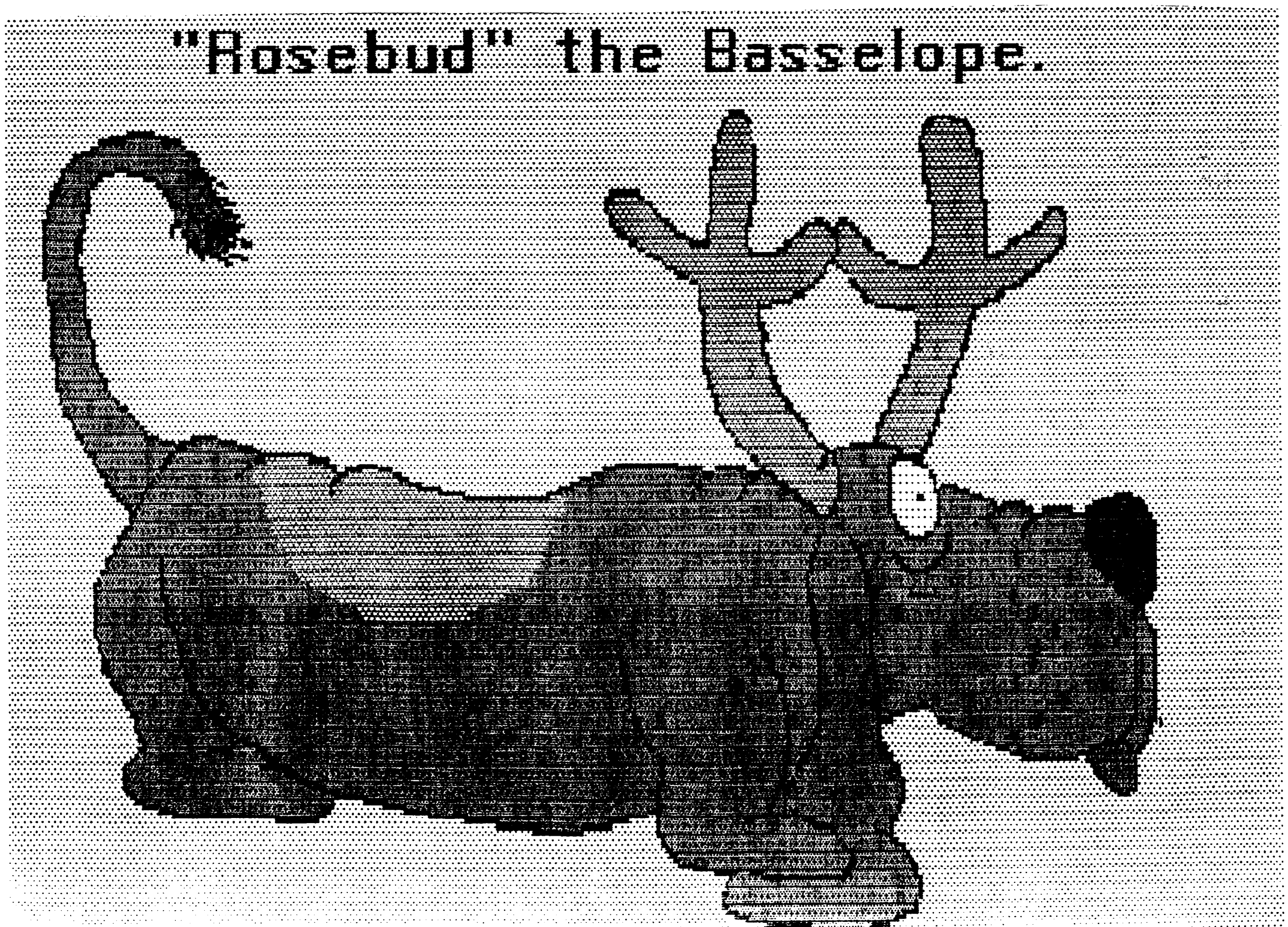
Educational Applications

Bakar Mudin of Weetangera is interested in contacting any other members using their Amiga for educational purposes (9-10 age group). Telephone - 549521 (H).

Modems

John Maher has been negotiating with a modem distributor to make a bulk purchase of modems at quite good prices. Anyone interested in buying a modem, to gain access to all the free information and public domain software on our bulletin board, should phone John on 974463 (H). Possible prices and modems are:

Cicada 300 (300 baud)	$ 99
Cicada 312 (300, 1200/75 baud)	$249
Nice Modem 1 (300, 1200/75, CCIT, BELL)	$279
Nice Modem 2 (300, 1200/75, 1200/1200, CCIT, BELL, Auto-Answer, Auto-Dial, Auto-Baud-Rate-Selection)	$699



BECAUSE

36 Ambalindum St.
HAWKER A.C.T. 2614
Ph. (062) 54 6033

Bulletin of the
Canberra Amiga
Users Society